Fakultät für
Informatik und Mathematik

Bachelorarbeit

über das Thema

Entwurf und Implementierung einer Chrome Extension zur automatischen Anreicherung von Webseiten mit kulturellen Inhalten

Autor: Mathias Möller
moellerm@fim.uni-passau.de

Prüfer: Prof. Dr. Granitzer

Abgabedatum: 31.08.2015

I Kurzfassung

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

II Inhaltsverzeichnis

III Abbildungsverzeichnis

1 Einleitung und Motivation

Suchmaschinen gehören heutzutage zu den meistbesuchten Seiten im Internet. Sie sind der Grundstein für den Zugriff auf Informationen im Internet [BH00]. Google[1] ist mit über 3 Milliarden Suchanfragen pro Tag [Lev] nicht nur die beliebteste Suchmaschine, sondern auch die meist besuchte Seite im Internet. Auch die chinesische Suchmaschine Baidu[2] ist in den oberen fünf Plätzen vertreten [AI]. Um so mehr sie an Bedeutung gewinnen, umso größer ist der Bedarf an bessern Suchmaschinen [Law00].

Doch der Funktionsumfang klassischer Suchmaschinen ist begrenzt. Sie können den Kontext des Users in ihren Anfragen nicht miteinbeziehen. Weiterhin verlangen sie vom Nutzer, seine gegenwärtige Arbeit einzustellen, die Webseite der Suchmaschine aufzurufen und eine Suchanfrage zu formulieren. So wird die Konzentration auf den eigentlichen Arbeitsschritt gestört. Im Schnitt sind nur 67% der Suchanfragen an Google erfolgreich [Kah]. Das heißt, in circa einem Drittel der Fälle bleibt der Aufruf der Suchmaschine ohne die erwarteten Erfolge. Zu der verlorenen Zeit addiert sich noch die Zeit, die der Nutzer braucht um sich seine Arbeit wieder ins Kurzzeitgedächtnis zu holen.

Die Interfaces der Suchmaschinen sind für Computer leichter zu bedienen als für menschliche Benutzer. Anfragen müssen auf die wichtigsten Schlüsselwörter reduziert werden und die Ergebnisse können maschinell deutlich besser verarbeitet werden als durch den Nutzer [BH99]. Auf der anderen Seite bestehen 62% der von Nutzern erstellen Suchanfragen nur aus ein bis zwei Wörtern [JSS00] und sind damit zu kurz. Kontextuelle Informationen werden dabei ausgelassen und die Anfragen sind oft mehrdeutig [BH99].

Um diesen Problemen entgegen zu wirken wurden neue Wege zur Informationsgewinnung entwickelt. Eine Möglichkeit sind so genannte Just-in-time Information Retrieval Agents (JITIR-Agents) [Rho00b]. Sie beobachten im Hintergrund den Kontext des Users und versuchen aus den so erhaltenen Informationen eine Suchanfrage an eine Datenbank oder ein Recommender-System zu schicken. Die gewonnenen Informationen werden dann möglichst unaufdringlich dem User dargestellt. Er kann sich nun entscheiden diese Informationen genauer zu betrachten oder mit seiner Arbeit fortzufahren. Die kognitive Belastung bleibt hierbei sehr gering. JITIR-Agents reduzieren auf diese Weise enorm den Aufwand Informationen zu finden [RM00].

Durch ihre Funktionsweise sind sie jedoch nicht so exakt wie klassische Suchmaschinen, da sie nur "erraten" können, was den User gerade interessiert. Wenn ein User einer genaue Vorstellung von der Suchanfrage oder den Ergebnissen hat, haben klassische Suchmaschi-

[1] `https://www.google.com/`
[2] `http://www.baidu.com/`

nen Vorteile gegenüber den JITIR-Agents [RM00].

Ein weiteres Problem des Internets ist es, dass die Standards, die die simple und leicht skalierbare Architektur des Netzwerks ermöglichen, den Einsatz von fortgeschritteneren Hypermedia-Technologien verhindern [Bou99]. Eine Möglichkeit, das Internet um Funktionen zu erweitern ist Web Augmentation (WA). Diaz [Día12] beschreibt WA als den Versuch, statt eine neue Technologie zu entwickeln, neue Funktionalität auf eine gerenderte Webseite zu setzen:

> In some sense, WA is to the Web what Augmented Reality is to the physical world: layering relevant content/layout/navigation over the existing Web to customize the user experience. [Día12]

Solche WA-Tools interagieren mit einem Web Server, Http-Proxy oder dem Browser der Nutzers, um so Inhalte oder Navigationselemente direkt in die angezeigte Webseite einzufügen. Auf diese Weise erlauben sie es, die limitierten Möglichkeiten des World Wide Webs zu Gunsten des Nutzers anzureichern [And97].

Ein Beispiel für ein solches WA-Tool sowie für ein JITIR System ist die EEXCESS Chrome Extension[3]. Sie analysiert die aktuell besuchten Seiten des Nutzers und schlägt ihm auf Grund der erlangten Daten am Rand des Browser-Fensters weiterführende Quellen aus der Europeana-Datenbank[4] vor.

Ziel dieser Bachelorarbeit ist es, auf Basis der EEXCESS Extension eine Browser Extension zu entwerfen und zu implementieren, welche eine Webseite in einzelne Paragraphen aufteilt und zu jedem dieser Paragraphen kulturelle Inhalte der Europeana-Datenbank vorschlägt. Dabei soll das Design der Extension helfen, die bei der in Kapitel 2 erläuterten Evaluation der EEXCESS Extension aufgetauchten Probleme zu beheben und Schwachstellen zu verbessern.

Um dieses Ziel umzusetzen, wird unter anderem auf Interactive Query Construction, im Speziellen auf Graphic Query Construction zurückgegriffen. Diese Methoden erlauben es dem User, die anfangs noch groben und kurzen Suchanfragen iterativ zu verfeinern [GW99], in dem er neue Schlüsselwörter zur Suche hinzufügt [Rut03]. Im Fall von Graphic Query Construction wird diese Verfeinerung über eine graphische Benutzeroberfläche durchgeführt.

Im anschließenden Teil dieser Arbeit wird auf vergleichbare Technologien eingegangen und ihre Gemeinsamkeiten und Unterschiede zu diesem Projekt. Nachfolgend wird das

[3] `http://eexcess.eu/results/chrome-extension/`
[4] `http://www.europeana.eu/portal/`

Konzept und die Implementierung des Projekts beschrieben und das Ergebnis evaluiert. Der sechste Teil der Ausarbeitung widmet sich Möglichkeiten für zukünftige Projekte und Verbesserungen. Zuletzt wird ein Fazit gezogen und die Ergebnisse der Arbeit analysiert. Der Sourcecode der im Zuge dieser Arbeit entwickelten Extension befindet sich unter `https://github.com/mathiasmoeller/eexcessredesign`.

2 Related Work

Rhodes [RM00] definiert JITIR-Agents als eine Klasse von Programmen, die dem User weiterführende Informationen basierend auf seinem lokalen Kontext anzeigen. Dabei beschränkt sich der überwachte Kontext meist auf die virtuelle Umgebung des Users, wie E-Mail, Webseiten und geöffnete Dokumente. Als Kerneigenschaften von JITIR-Agents nennt Rhodes Selbstständigkeit, die Fähigkeit Informationen in einer leicht zugänglichen und gleichzeitig unaufdringlichen Weise darzustellen und das Bewusstsein über den Kontext des Users [Rho00b]. Das im Zuge dieser Bachelorarbeit entworfene Programm (im Folgenden als Jarvis bezeichnet) erfordert vom Nutzer die explizite Aufforderung um nach Informationen zu suchen. Gründe für diese Entscheidung werden im dritten Kapitel näher betrachtet. Trotz dieses Widerspruchs zu Rhodes Definition lässt sich Jarvis am besten mit dieser Klasse von Programmen vergleichen. Warum diese Klassifizierung zutrifft wird im folgenden Abschnitt beschrieben.

2.1 Unterschiede und Gemeinsamkeiten zu JITIR-Agents

Studien haben gezeigt, dass schon eine kleine Steigerung des Aufwands, der betrieben werden muss um eine Aufgabe zu erfüllen, dazu führen kann, dass man die Aufgabe gar nicht erst ausführt [RM00]. Laut Miller reicht für die meisten Aufgaben eine Antwortverzögerung von mehr als zwei Sekunden aus, um die Nutzungshäufigkeit des dazugehörigen Programms zu vermindern [Mil68]. Längere Zeitintervalle erschweren es, den Kontext der gerade ausgeführten Aufgabe und die übergeordneten Aufgaben im Kurzeitgedächtnis zu behalten. Nun wird der Fall betrachtet, dass das Lesen einer Webseite unterbrochen wird, um eine Suche mit einer Suchmaschine durchzuführen. Die eigentliche Aufgabe behält der Nutzer im Kurzzeitgedächtnis. Je länger die Suche dauert und je mehr er sich dazu von seiner eigentlichen Arbeit distanzieren muss, desto schwerer wird es wieder zur Hauptaufgabe zurück zu kehren. Wenn der Exkurs zur Suchmaschine schwerer wiegt ist als die Güte der erwarteten Resultate wird die Suche nicht durchgeführt [RM00].

Dieses Problem wird versucht mit JITIR-Agents zu beheben. Durch ihre proaktive Arbeitsweise muss der Nutzer seine Tätigkeit nicht mehr unterbrechen, sondern nur noch entscheiden ob er weiter Informationen sehen möchte oder nicht [RM00]. Beim Entwurf von Jarvis wurde entschieden, dass das Programm nicht völlig eigenständig nach Informationen sucht, sondern nur die Webseite in seine Paragraphen aufteilt. Der Nutzer kann dann entscheiden, ob er zu einem Paragraphen eine Suche durchführen möchte und diese dann per Klick starten. Wie bei einem JITIR-Agent wird die Suchanfrage automatisch aus den gewonnenen Kontextinformationen generiert. Da die Suche innerhalb weniger Millisekunden ausgelöst werden kann und sich der Nutzer dazu nicht von seiner eigent-

lichen Aufgabe distanzieren muss, bleibt die kognitive Belastung sehr gering. Allerdings entsteht auch so der Nachteil, den Jarvis mit JITIR-Agents gemein hat: Sie nutzen alle gefunden Informationen für ihre Suchanfragen und können nicht zwischen relevanten und unwichtigen Suchwörtern unterscheiden [Rho00a]. Automatisch gebaute Suchanfragen sind deshalb weniger exakt als Menschen-generierte [RM00]. Um dem entgegen zu wirken hat der User von Jarvis im Nachhinein noch die Möglichkeit, die Suche anzupassen und erneut abzuschicken.

Die zweite von Rhodes beschriebene Eigenschaft von JITIR-Agents, die Fähigkeit Informationen in einer leicht zugänglichen und gleichzeitig unaufdringlichen Wiese darzustellen, hat auch beim Entwurf von Jarvis eine bedeutende Rolle gespielt. Für die Darstellung der Ergebnisse muss ein Mittelwert gefunden werden. Die zusätzlichen Elemente sollen den User nicht unnötig ablenken, allerdings will man die Seite um möglichst reichhaltige Informationen erweitern [Rho00a]. Wie diese Problematik im Falle von Jarvis gelöst wurde wird im Implementierungs-Teil dieser Arbeit beschrieben.

Jarvis analysiert die geöffnete Webseite und teilt diese in Paragraphen ein. Wie JITIR-Agents ist er sich also über den lokalen Kontext des Users bewusst und wertet diesen aus. Die dritte Voraussetzung für JITIR-Agents ist somit erfüllt.

Trotz der eingeschränkten Selbstständigkeit lässt sich Jarvis folglich am besten mit denen von Rhodes beschriebenen Programmen vergleichen. Auf diese Erkenntnis aufbauend wird nachfolgend Jarvis mit existierenden Implementierungen von JITIR-Agents verglichen.

2.2 Vergleich mit existierenden JITIR-Agents

Es wurden einige JITIR-Agents in der Vergangenheit implementiert, die zwar aus technischer Hinsicht nicht mehr aktuell sind, die sich jedoch durch ihre genaue Untersuchung und Auswertung gut Vergleichen lassen.

2.2.1 Remembrance Agent

Der Remembrance Agent (RA) ist ein in den UNIX Texteditor EMACS-19 integriertes Programm, welches eine Liste von Dokumenten anzeigt, die für den Nutzer interessant seien könnten [Rho00b]. Die Informationen für die Vorschläge bezieht der RA aus der gerade geöffneten Datei. Wie Jarvis teilt er das Dokument in Bereiche ein, die er danach analysiert. Allerdings sind die Bereiche in diesem Fall keine Paragraphen sondern verschieden große Bereiche in denen er sucht. Von der Position des Cursors ausgehend untersucht er die nächsten zehn, die nächsten 50 und die nächsten 1000 Wörter des Dokuments und sucht zu jedem passende Vorschläge [RS96]. Die Vorschläge werden von einem

Information Retrieval Programm im Hintergrund generiert, welches die Suchanfragen an verschiedene, vom Nutzer konfigurierbare Datenbanken schickt. Genannte Datenbanken sind persönliche Email-Verzeichnisse, persönliche Notizen oder Datenbanken mit wissenschaftlichen Arbeiten.

Die gefundenen Ergebnisse werden dem Nutzer dann am unteren Fensterrand dargestellt. Je nach Art der Quellen werden verschiedene Informationen angezeigt. So werden zum Beispiel bei wissenschaftlichen Arbeiten der Autor, das Datum und der Titel angezeigt und bei Zeitungsartikeln nur Herausgeber, Überschrift und Datum [RM00]. Dieses Design ermöglicht eine unaufdringliche Darstellung der Vorschläge, die leicht überblickt werden kann aber den Nutzer nicht von seiner Arbeit ablenkt [RS96]. Per Klick kann sich der Nutzer die Schlagwörter anzeigen lassen, die für das jeweilige Ergebnis verantwortlich sind [Rho00b]. Mit einer Tastenkombination und der Zeilennummer des Vorschlags kann er sich das dazugehörende, vollständige Dokument anzeigen lassen. Weiterhin hat der Nutzer die Möglichkeit, eine eigene Suchanfrage einzugeben um so explizit nach Inhalten zu suchen [RS96].

Mit Hilfe einer Nutzer-Evaluation konnte gezeigt werden, dass der RA nicht nur eine Alternative zu klassischen Suchmaschinen darstellt, sondern in vielen Bereichen sogar besser abscheidet. Durch die Nutzung fanden die Tester mehr relevante Dokumente zum geforderten Themengebiet und die anschließende Umfrage schnitt der RA besser ab als die der Kontrollgruppe zur Verfügung gestellte Suchmaschine [RM00].

Der RA verwendet zwar einen anderen Kontext als Jarvis, das Design folgt jedoch ähnlichen Prinzipien. Wie Jarvis wird ein Ramping Interface zur Anzeige der Informationen genutzt und der User hat Zugriff auf die Terme der Suchanfrage und kann diese manipulieren. Dies geschieht jedoch ohne eine graphische Darstellung sondern durch Eingabe von Suchbegriffen.

2.2.2 Watson

Watson ist ein weiteres Beispiel für JITIR-Agents[5]. Er überwacht die Benutzung von Textverarbeitungs- und Textdarstellungsprogrammen wie Microsoft Word, Microsoft Internet Explorer oder Netscape Navigator [BH99]. Dazu werden sogenannte Application Adapter benutzt, die sich mit der jeweiligen Software verbinden um so an den Inhalt der angezeigten oder bearbeiteten Dokumente zu gelangen [BH00]. Die erlangten Informationen werden an Watson weitergegeben, welcher versucht eine passende Quelle aufgrund dieser Daten auszuwählen. Ein weiterer Prozess versucht zu erkennen, ob der Nutzer Be-

[5]Die Entwickler beschreiben das System als Information Management Assistant, die Bedeutung ist jedoch ähnlich.

darf an zusätzlichen Informationen hat. Trifft das zu, wird eine Anfrage an die ausgewählte Quelle/die ausgewählten Quellen geschickt. Die Ergebnisse werden dann gruppiert und in einem separaten Fenster angezeigt [BH99].

Auch bietet Watson die Möglichkeit, eine Suchanfrage direkt einzugeben. Die Anfrage des Nutzers wird daraufhin mit der automatisch erstellten, kontextabhängigen Anfrage kombiniert um so möglichst relevante Ergebnisse zu liefern [BH00]. Weiterhin werden atomare Bausteine, wie Adressen erkannt. Dem Nutzer wird dann ein Knopf angezeigt, über den er zu einer passenden Darstellung des Bausteins gelangt. Im Falle einer Adresse würde er zum Beispiel zu einer Karte weitergeleitet werden, in der die Adresse angezeigt wird.

Evaluationen haben gezeigt, dass die von Watson vorgeschlagenen Dokumente in fünf von zehn Fällen relevant waren. Die Kontrollgruppe, welche die Suche manuell mit Alta Vista[6] durchführen mussten (Watson benutzte als Quelle auch Alta Vista), kamen im Schnitt auf nur drei relevante Ergebnisse. Schlussendlich erzielte Watson bessere Ergebnisse als die Kontrollgruppe in 15 von 19 Fällen [BH99].

Wie beim RA kann auch bei Watson der User die Suchanfrage durch Eingeben von neuen Termen manipulieren. Watson kombiniert die Anfrage des Users darauf hin jedoch noch mit den verfügbaren Kontextinformationen. Der Bereich in dem diese Informationen gesammelt werden erstreckt sich bei Watson, anders als bei Jarvis oder beim RA, über mehrere Programme hinweg. Somit steht hier die größte Menge an kontextrelevanten Daten zur Verfügung. Die Ergebnisse werden anders als bei den meisten JITIR-Agenten jedoch außerhalb des aktuellen Fensters in einem separaten Fenster angezeigt.

2.2.3 EEXCESS Chrome Extension

Die von EEXCESS[7] entwickelte Chrome Extension[8] benutzt JITIR Mechanismen um kulturelle Inhalte der Europeana Datenbank für den Nutzer leichter zugänglich zu machen. Die Software analysiert die aktuelle Webseite oder gegebenenfalls einen selektierten Paragraphen innerhalb dieser Seite und die gewonnenen Daten werden in Form einer Anfrage an Europeana geschickt [SSG14]. Die personalisierten Resultate werden dem User am Rande des Browserfensters in einem separierten Bereich angezeigt. Bei der Entwicklung von Jarvis wurde entschieden, die Resultate direkt beim jeweiligen Paragraphen anzuzeigen, um die Verbindung zwischen Quelle und Ergebnis deutlicher zu machen.

Durch die automatische Generierung der Suchanfragen sind diese nicht immer zielführend,

[6] `http://de.wikipedia.org/wiki/AltaVista`
[7] `http://eexcess.eu/`
[8] `http://eexcess.eu/results/chrome-extension/`

wie sich bei einer Evaluation ergab [SSL+15]. Da selbst die besten Heuristiken den Kontext des Benutzers nicht immer korrekt auswerten können (zum Beispiel wenn der Benutzer den Kontext gerade gewechselt hat), kommt bei Jarvis Interactive Query Construction zum Einsatz. Das ermöglicht es dem Nutzer, die generierten Anfragen nach seinen Wünschen anzupassen und die Suche neu durchzuführen.

2.3 Interactive Query Construction

Ein System welches Interactive Query Construction verwendet, ist das von Russel et al. entwickelte NITELIGHT [RSBS08]. Es unterstützt Nutzer bei der Erstellung und Anpassung von Suchanfragen auf Basis der SPARQL [PS] Abfragesprache. Das in Java entwickelte Programm erlaubt es dem Nutzer über eine graphische Oberfläche interaktiv semantische Anfragen zu bauen. Dazu werden ihm graphische Elemente zur Verfügung gestellt, die auf Konstrukten der Abfragesprache basieren [RSBS08]. Die in Jarvis verwendete Interactive Query Construction ist stark vereinfacht und ermöglicht es dem Nutzer lediglich, die automatisch generiert Anfrage zu erweitern oder irrelevante Schlüsselwörter zu entfernen. Jedoch ermöglicht die Nutzung dieser Methodik eine leichte Erweiterbarkeit um Funktionen zur Verbesserung der Ergebnisgüte wie zum Beispiel Relevance Feedback. Darauf wird in Kapitel 6 genauer eingegangen.

2.4 Zusammenfassung

Um Jarvis zu effizient wie möglich und so benutzerfreundlich wie möglich zu gestalten, wurde versucht aus den Stärken und Schwächen vergleichbarer Technologien zu lernen. Wie die EEXCESS Extension sendet Watson seine Anfragen an Europeana. Ähnlich zum RA teilt Jarvis das Dokument in verschiedene Bereiche ein und analysiert diese getrennt von einander. Wie auch die anderen Technologien bietet Jarvis die Möglichkeit an die initiale Suchanfrage zu ändern. Dazu wird jedoch ähnlich zu NITELIGHT Interactive Query Construction benutzt um den Prozess so einfach wie möglich zu gestalten. Um die kognitive Belastung zu reduzieren und um den Zusammenhang zwischen den Paragraphen und den Ergebnissen eindeutig darzustellen.

3 Konzept

Wie auch bei der von EEXCESS entwickelten Extension fiel auch bei Jarvis die Wahl auf eine Chrome Extension. Google Chrome hat mit 40% den größten Marktanteil unter Web Browsern und durch die Ähnlichkeit der Extension Architektur von Chrome zu anderen Browsern kann die Anwendung im Nachhinein leicht auf diese übertragen werden [SSG14]. Per "JavaScript Injection" kann in diesen Architekturen das Aussehen und Verhalten der Seite verändert und neue Funktionalität hinzugefügt werden [SSG14].

Beim Entwurf von Jarvis wurde sich am Design von JITIR-Agents orientiert. Trotzdem sucht Jarvis nicht proaktiv nach Ergebnissen, sondern erst nach einer Interaktion der Benutzerin. Zum einen entstehen durch die Nutzung der Europeana API technische Limitierungen. Für jeden API-Key lässt Europeana pro Tag nur 10.000 Anfragen zu [VK]. Dieser API-Key wird von allen installierten Jarvis Extensions genutzt. Da in jeder geöffneten Webseite für jeden gefundenen Paragraphen eine eigene Anfrage an die Europeana API gesendet werden würde, wäre dieses Limit ab einer gewissen Anzahl von Nutzern schnell überschritten. Zum anderen kann die Nutzerin so entscheiden, ob er zum jeweiligen Paragraphen weitere Informationen erhalten möchte oder nicht. Möchte er das nicht, wird seine Aufmerksamkeit auch nicht durch das erscheinen von Ergebnissen gestört. Der letzte Grund, der zu dieser Entscheidung geführt hat, ist, dass das automatische Absenden aller Suchanfragen Rechenleistung benötigen würde, wodurch der Browser verlangsamt werden würde.

Der Entwurf der Extension lässt sich in fünf Ziele einteilen: 1. Die Erstellung der Suchanfrage 2. Die Anzeige der Ergebnisse 3. Die Erklärung der Suchanfragen Generierung 4. Das Anpassen der Suchanfrage durch die Benutzerin 5. Die Verbesserung der Ergebnisgüte. Auf diese Ziele wird im folgenden genauer eingegangen.

3.1 Erstellung der Suchanfrage

Um eine Suchanfrage abschicken zu können müssen erst Schlüsselwörter gefunden werden. Das geschieht entweder durch das analysieren des Paragraphen oder durch Hinzufügen von Wörtern durch die Benutzerin. Die Schlüsselwörter müssen daraufhin in eine Suchanfrage umgewandelt werden, die die Sprache der angesprochenen Datenbank verwendet. Im Falle von Europeana ist das zum Beispiel eine boolesche Abfragesprache. Zur eigentlichen Suchanfrage werden noch Suchparameter, wie maximale Anzahl der erwarteten Ergebnisse, hinzugefügt. Ist das geschehen wir sie abgeschickt.

3.2 Anzeige der Ergebnisse

Nachdem eine Anfrage an die Datenbank gesendet wurde und die Ergebnisse beim Client angekommen sind, müssen diese dargestellt werden. Die Nutzerin soll leicht erkennen können, welche der ihm angezeigten Elemente aus der ursprünglichen Webseite stammen und welche durch die Extension hinzugekommen sind. Aus diesem Grund wurde bei Jarvis ein einheitliches und buntes Design gewählt. So setzt sich die Anwendung deutlich von den meisten Webseiten ab. Auch soll eine enge Bindung zwischen den Ergebnissen und dem dazugehörigen Paragraphen geschaffen werden. Um das zu erreichen, werden die durch die Paragraphen-Erkennung gefundenen Textpassagen hervorgehoben und die Ergebnisse direkt bei den jeweiligen angezeigt.

Weiterhin ist es wichtig, einen Mittelwert zu finden, was die Menge der angezeigten Informationen betrifft. Zu viele Informationen auf einmal können die Nutzerin ablenken. Auf der anderen Seite ist es das Ziel der Userin eine möglichst reichhaltige alternative Informationsquelle zu liefern [Rho00a]. Rhodes empfiehlt hierfür ein Ramping Interface [RM00]. Hierbei werden die Informationen in Stufen aufgeteilt. Jede Stufe liefert etwas mehr Informationen als die vorherige. Auf diese Weise kann die Benutzerin entscheiden ob sie mehr Informationen sehen möchte und daraufhin auf die nächste Stufe gehen oder nicht.

Jarvis zeigt nach einer erfolgreichen Suche zunächst nur die Anzahl der gefundenen Ergebnisse an. Diese Anzeige ist aufgeteilt in Anzahl der textuellen Ergebnisse, der gefundenen Bilder sowie der gefundenen Audio- und Videoquellen. Durch Auswählen einer der drei Kategorien werden in einem sich über dem Paragraphen öffnenden Fenster eine Liste mit Titel und Vorschaubild der Ergebnisse angezeigt. Die Liste ist absteigend nach der Relevanz sortiert, die Europeana in Form eines Relevance Scores mitliefert. Durch ändern des Tabs kann hier zwischen den Kategorien gewechselt werden. Durch einen Klick auf den Titel oder das Bild eines Listeneintrages kann die Userin die entsprechende Quelle aufrufen. Bewegt sie bei Text- oder Audio-/Videoquellen den Cursor über das Miniaturbild eines Ergebnisses, werden ihr weitere Informationen angezeigt. Da bei Bildquellen die Metadaten nicht so wichtig sind wie das Bild selber, werden sie in einer Gitter-Liste angezeigt. Die Listen-Elemente bestehen nur aus einem Vorschaubild sowie dem Titel. Durch klicken gelangt die Benutzerin zur Quelle des Bildes. Weitere Informationen werden nicht angezeigt, da sie für die Entscheidung die Quelle aufzurufen in den meisten Fällen unwichtig sind.

3.3 Erklärung der Suchanfragen Generierung

Für die Userin soll es einfach zu erkennen sein, wie die Ergebnisse zustande gekommen sind. Dafür ist es wichtig, dass sie sieht aus welchen Suchbegriffen die Anfrage zusammengesetzt wurde. Nach dem Auslösen der Suchfunktion der Extension werden die Schlagwörter, die aus diesem Paragraphen extrahiert wurden, über diesem angezeigt. Diese Anzeige erlaubt auch die Manipulation der Suchanfrage. Weiterhin werden die gefundenen Schlagwörter im Text hervorgehoben. So soll gewährleistet werden, dass die Entstehung der Suchanfrage offensichtlich ist und die Nutzerin intuitiv erkennt, dass die Anfrage über die genannten Elemente angepasst werden kann.

3.4 Anpassen der Suchanfrage durch die Benutzerin

Da automatisch generierte Suchanfragen nicht immer die gewünschten Ergebnisse liefern, kann die Nutzerin die Suchanfrage selber erstellen oder die generierte manipulieren. Dazu kann sie Wörter aus dem Text hinzufügen oder eigene eingeben. Aus der bestehenden Anfrage kann sie unpassende oder mehrdeutige Suchbegriffe entfernen. Dazu wird die im vorigen Absatz beschriebene Anzeige der Suchbegriffe genutzt. Durch diese Interaktionsmöglichkeiten soll das Programm so intuitiv wie möglich gestaltet werden um Nutzer mit unterschiedlichen Erfahrungsleveln anzusprechen. Die beschriebenen Funktionalitäten können weiterhin für interaktive Elemente der Query Extension (siehe Kapitel 6) genutzt werden.

3.5 Verbesserung der Ergebnisgüte

Durch den Einsatz von Methoden des maschinellen Lernens sowie von Query Extension soll die Relevanz der Ergebnisse verbessert werden [Rut03]. Für diesen Anwendungsbereich geeignete Techniken sind Automatic Query Extension[9] (z.B. das Erstellen von Suchprofilen der User [BH00]) und Interactive Query Extension (z.B. Relevance Feedback [Har88]. Bei Query Extension werden, mit oder ohne Benutzerinteraktion, zusätzliche Begriffe zur Suchanfrage hinzugefügt. Da von Usern erstellte Suchanfragen meistens nur sehr kurz [Rut03] und dadurch oft nicht eindeutig sind, kann so eine Steigerung der Ergebnisgüte erreicht werden. Durch die hinzugefügten Suchwörter können falsche Ergebnisse, die durch andere Bedeutungen der alten Suchwörter entstanden sind, heraus gefiltert werden.

Ein Beispiel für Automatic Query Extension ist das Erstellen von Suchprofilen. Das Programm versucht dabei vom Verhalten der Userin zu lernen. Dazu werden Informationen über alle Suchvorgänge hinweg gespeichert. Diese Informationen können durch das Bewerten von Seiten durch den Nutzer gesammelt werden oder aus den Seiten extrahiert werden,

[9] Automatic Query Extension ist hierbei ein Beispiel für maschinelles Lernen.

bei denen die Nutzerin ein Lesezeichen gesetzt hat [BH00]. Die gewonnenen Informationen werden dann in Schlüsselwörter umgewandelt und an die Suchanfragen angehängt.

Interactive Query Expansion erlaubt der Userin, die initiale Anfrage noch zu erweitern [Har88]. Bei Relevance Feedback zum Beispiel werden der Benutzerin die Ergebnisse ihrer ersten Suchanfrage präsentiert. Sie kann daraufhin einzelne Quellen als relevant oder irrelevant bewerten. Auf Grund dieser Bewertung wird die ursprüngliche Anfrage um positive und negierte Terme erweitert [BH00].

Da das den Rahmen dieser Arbeit jedoch sprengen würde, wird auf die Integration dieser Methoden nur in Kapitel 6 Bezug genommen.

3.5.1 Speicherung von guten Suchanfragen für jeden Paragraphen

3.5.2 Bewertung von Ergebnissen

TODO: - Speichern von guten Queries pro Paragraph - bewertung von queries (vie viele ergebnisse wurden angeschaut, wurden überhaupt ergebnisse angeschaut) - bewertung der ergebnisse - lernen was gute queries sind

4 Implementierung

Im nachfolgenden Kapitel wird beschrieben, welche Technologien eingesetzt wurden und wie die Architektur der Anwendung aussieht. Des weiteren werden die wichtigsten AngularJS Services und Direktiven vorgestellt, die für die Anwendung implementiert wurden und es wird beschrieben, wie die Benutzeroberflächen umgesetzt worden sind.

4.1 Verwendung von AngularJS für alle Komponenten des Plugins

Für die Implementierung von Jarvis wurde sich für AngularJS[10] entschieden. Das von Google entwickelte JavaScript MVC-Framework bietet einige Vorteile, die Webentwicklung im Allgemeinen und die Entwicklung dieser Extension erleichtern.

AngularJS erlaubt es eigene Direktiven zu entwerfen. Direktiven beschreiben wie ein HTML Element dargestellt werden soll und wie es (z.B. auf Benutzerinteraktionen) reagieren soll [JMM15]. Sie können dann wie HTML Tags oder Attribute genutzt werden. So wird eine hohe Wiederverwendbarkeit von Komponenten erzeugt. Genutzt wurde dieses Feature um eine Direktive zu entwickeln, mit der die Paragraphen der Webseite "verpackt" werden. Die neue Direktive enthält die Funktionen um eine Suchanfrage zu bauen und zu senden und die graphischen Elemente um die Resultate anzuzeigen.

Weiterhin befindet sich in den meisten Frameworks das Markup in einem Template. Dieses Template wird dann in HTML Code kompiliert und in das Document Object Model (DOM) geladen [JMM15]. Das Markup der AngularJS Application befindet sich innerhalb des HTML-Dokuments. Dadurch ist es möglich, dass AngularJS das Markup erst auswertet, nachdem es in das DOM geladen wurde [JMM15]. Von den Vorteilen die dadurch entstehen ist einer für die Entwicklung von Jarvis besonders nützlich: Da AngularJS die Seite erst nach dem Laden auswertet, ist es einfach, AngularJS Module in existierende Webseiten zu integrieren [JMM15]. Mit Jarvis möchte genau das erreicht werden. Zu einer bestehenden Seite soll Funktionalität hinzugefügt werden. Zusammen mit JavaScript Injection kann man auf diese Weise innerhalb des Client-Browsers AngularJS Logik in eine beliebige Seite integrieren, ohne die eigentliche Implementierung der Seite zu verändern.

Der Modul-basierte Aufbau von AngularJS Anwendungen gestattet außerdem eine leichte Erweiterbarkeit und Austauschbarkeit der Komponenten. Einzelne Dienste, wie bei Jarvis die Extrahierung der Schlüsselwörter oder die Kommunikation mit Europeana, sind in Module oder Services verpackt und können so einfach durch andere ersetzt werden. So kann man die Extension leicht auf andere Suchmaschinen umbauen oder andere Algorithmen zur Analyse der Paragraphen verwenden.

[10] `https://angularjs.org/`

Unabhängig von der Entwicklung von Jarvis bietet AngularJS einen weiteren großen Vorteil: Two way data-binding [JMM15]. Im Gegensatz zu anderen Frameworks oder zu reinem JavaScript, wo man manuell jede Änderung des Models an die View propagieren muss und anderes herum, übernimmt AngularJS diese Aufgabe. Ändert sich das Model, wird automatisch das jeweilige DOM Element angepasst. Interagiert der User mit einem Element, werden Änderungen sofort an das dahinter stehende Model weitergeben. Das reduziert den benötigten Code und erlaubt eine schnelle und saubere Entwicklung von Anwendungen.

4.2 Architektur der Extension

Wie in Abbildung 1 zu sehen ist, ist Jarvis in vier AngularJS Module unterteilt: a) Jarvis für die Content-Skripte, b) JarvisBG für die Background-Skripte, c) JarvisPopup für die Browser-Action, und d) JarvisSettings für die Options-Page. Im folgenden Absatz beschreibt Jarvis das Modul der Content-Skripte und nicht die gesamte Extension.

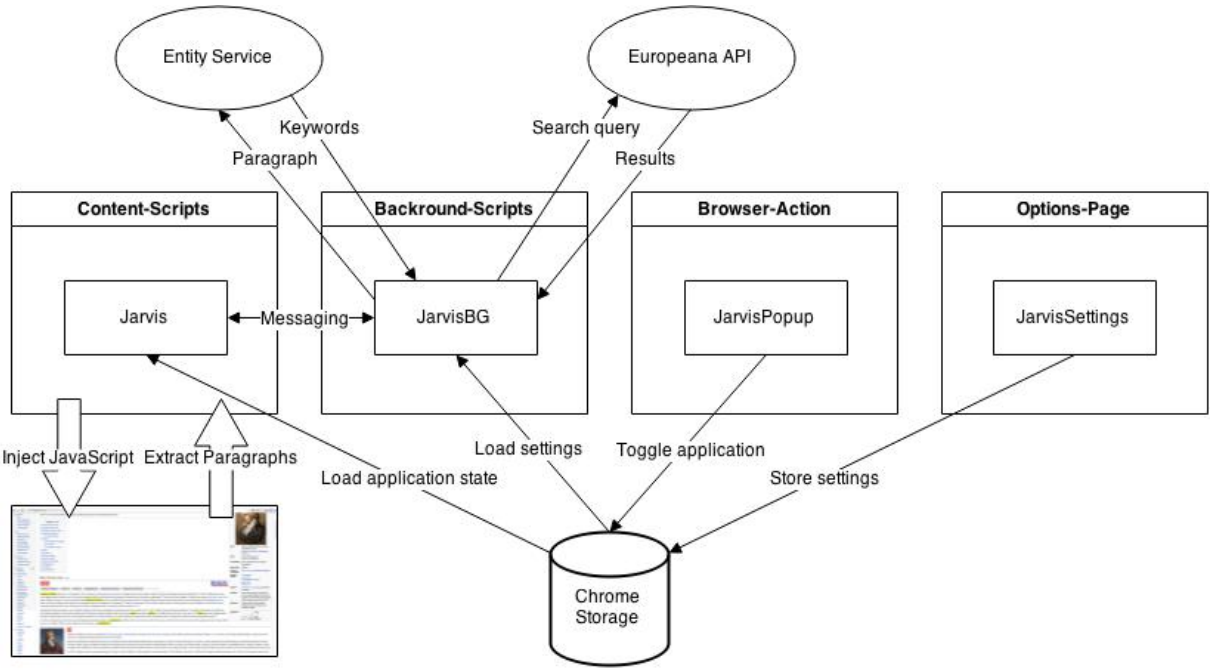

Abbildung 1: Kommunikation der AngularJS Komponenten untereinander

Jarvis ist das Modul, welches in die jeweilige Webseite per JavaScript Injection eingefügt wird. Es extrahiert die Paragraphen aus dem Text und fügt die eigene Direktive (im Folgenden Paragraph-Directive genannt) in das DOM ein. Die extrahierten Paragraphen werden per Message Passing[11] an die Background-Skripte weitergegeben. Diese sind zu-

[11] `https://developer.chrome.com/extensions/messaging`

sammengefasst im Modul JarvisBG.

Dieses Modul ist zuständig für die Kommunikation mit den REST Endpunkten von Europeana und dem Entity Service. Im Gegensatz zu Jarvis, welches für jeden offenen Browser-Tab eine eigene Instanz hat, existiert das JarvisBG Modul nur ein mal (wie auch JarvisPopup und JarvisSettings). Jarvis schickt Nachrichten mit dem Namen des angeforderten Services sowie einer Callback-Methode an JarvisBG. Dieses führt die Anfrage aus und übergibt die Ergebnisse der Callback-Methode. Diese sorgt dann dafür, dass die Ergebnisse im Frontend richtig angezeigt werden.

JarvisPopup ist das Modul des Popup-Fensters, welches nach klick auf das Icon der Extension rechts des Adressfeldes angezeigt wird. Über das Popup wird der Zustand der Extension im aktuellen Tab gesteuert. Der User kann hier entscheiden, ob das Programm in diesem Tab angezeigt wird oder nicht. Der Zustand wird dann im Chrome Storage[12] gespeichert. Die Wahl für die Speicherung der Anwendungsdaten viel auf den Chrome Storage, da er einige Vorteile gegenüber der localStorage API[13] bietet. Zum einen können die Content-Skripte direkt darauf zugreifen ohne einen Umweg über die Background-Skripte gehen zu müssen. Weiterhin erlaubt Chrome Storage eine automatische Synchronisierung der Inhalte. Dadurch wird Jarvis sofort benachrichtigt, sobald sich der Zustand der Applikation ändert.

Der Nutzer hat die Möglichkeit, über die Options-Page[14] die verwendeten Services zu konfigurieren. Das dafür zuständige Modul ist JarvisSettings. Es speichert die Einstellungen im Chrome Storage und JarvisBG passt die REST-Anfragen entsprechend an.

4.3 Eigene Services und Direktiven

Um eine hohe Modularität und Austauschbarkeit zu gewährleisten sowie eine übersichtliche Architektur zu schaffen wurden einzelne Funktionalitäten in AngularJS Services und Direktiven ausgelagert. So können zum Beispiel die Services zur Kommunikation mit Europeana oder zum Finden der Schlüsselwörter einfach ausgetauscht werden um andere Algorithmen/Informations-Quellen zu benutzen. Die wichtigsten Services und Direktiven werden hier vorgestellt.

4.3.1 ParagraphDetectionService

Jarvis teilt die aktuelle Webseite in Paragraphen ein. In diesem Fall sind damit nicht HTML Paragraphen Tags (`<p>`) gemeint, sondern jegliche zusammenhängende Textpas-

[12] `https://developer.chrome.com/extensions/storage`
[13] `https://developer.mozilla.org/en-US/docs/Web/Guide/API/DOM/Storage#localStorage`
[14] `https://developer.chrome.com/extensions/optionsV2`

sage die größer als ein bestimmter Schwellwert ist. Der Algorithmus dafür stammt vom Lehrstuhl für Medieninformatik an der Universität Passau.

Um Paragraphen innerhalb der Seite zu finden wird das DOM mit einem TreeWalker[15] durchlaufen. Es werden alle Knoten herausgefiltert, deren Vorgängerknoten Script (`<script>`), Style (`<style>`) oder Noscript Tags (`<noscript>`) sind, sowie Knoten die keinen Text oder weniger als 40 Zeichen enthalten. Die restlichen Knoten werden nun einzeln betrachtet um zusammenhängende Paragraphen zu erkennen. Dazu werden jeweils die Nachbarknoten betrachtet. Gibt es einen Nachbarknoten der kein reiner Textknoten ist und sich ebenfalls in der Liste der gefundenen Paragraphen befindet so werden beide als zusammengehörig abgespeichert. Alle Knoten zu denen kein passender Nachbar gefunden wurde, die jedoch mehr als 100 Zeichen haben und mindestens einen ".", werden als einzelne Paragraphen gespeichert.

Daraufhin werden nochmal alle zusammengehörigen und einzelnen Paragraphen durchlaufen. Dabei wird der Vorgängerknoten jedes Elements betrachtet. Ist dieser ein Header Knoten so wird er als Überschrift des Paragraphen gespeichert. Jedes Element wird dann in die ParagraphDirective eingebettet.

Schlussendlich werden noch alle Anchor Tags (`<a>`) innerhalb der Paragraphen entfernt. Sie verhindern das Auswählen eines Schlüsselwortes aus dem Text per Doppelklick und werden deshalb durch AnchorDirectives ersetzt.

Verpackt ist dieser Algorithmus in einen AngularJS Service (ParagraphDetectionService), der im Modul Jarvis registriert ist. Nach dem Laden der Extension führt er die Paragraphen-Suche aus und stellt Funktionen zum abrufen der gefunden Textpassagen bereit.

4.3.2 KeywordService

Um aus den gefundenen Paragraphen die wichtigen Schlüsselwörter zu extrahieren wird ein REST-Service genutzt (im Konzept Entity Service genannt), der auch vom Lehrstuhl für Medieninformatik an der Universität Passau bereitgestellt wurde. Dieser Service basiert auf DBpedia Spotlight[16], einem System zum automatischen Annotieren von DBpedia[17] Entities in Texten [DJHM13].

DBpedia Spotlight ist ein Open Source Projekt, das einen REST-Service entwickelt hat, welcher Interfaces für Phrase Spotting (Erkennen von Ausdrücken, die annotiert werden

[15] `http://www.javascriptkit.com/dhtmltutors/treewalker.shtml`
[16] `http://spotlight.dbpedia.org/`
[17] `http://wiki.dbpedia.org/`

können) und Disambiguierung (Assoziieren von Phrasen mit DBpedia Entities) anbietet [DJHM13]. Dabei analysiert der Service einen Text in vier Phasen:

Während des Phrase Spottings werden die Phrasen innerhalb des Textes gesucht, die zu einer DBpedia Ressource passen könnten. Dazu durchläuft ein String-Matching-Algorithmus den Text. Als Suchmaske wird ein Set aus Labels genutzt, welches aus DBpedia Ressourcen gewonnen wurde [MJGSB11]. Danach folgt die Candidate Selection: Den gefundenen Kandidaten werden mögliche Disambiguierungen aus dem DBpedia Lexikalisierungs Datensatz zugewiesen. Im nächsten Schritt, der Disambiguation, wird der Kontext - der ursprüngliche Text - dazu genutzt die am besten passenden Disambiguierungen für jeden Kandidaten zu finden. Dazu wird aus den DPpedia Ressourcen ein Vektor Space Model gewonnen und die möglichen Disambiguierungen werden nach der Ähnlichkeit ihres Kontext-Vektors und dem Kontext des Ursprungstextes bewertet. Der letzte Schritt bietet dem User die Möglichkeit den Prozess zu personalisieren. Er kann zum Beispiel Listen mit erlaubten oder verbotenen Entities übergeben, die in der Candidate Selection genutzt werden [MJGSB11]. Zum Schluss werden die finalen Kandidaten und die jeweils passende Disambiguierung zurück gegeben.

Die erhaltenen Kandidaten werden dann als Schlüsselwörter für die Suche bei Europeana genutzt. Allerdings sind diese Kandidaten nicht immer Wörter oder Phrasen, die direkt im Text vorkommen. So wird zum Beispiel das Wort “Marbach” nicht genauso wieder zurück gesendet, sondern der dazugehörige Kandidat wäre “Marbach am Neckar”. So wird zwar eine höhere Genauigkeit erzielt, jedoch können diese Kandidaten dann nicht wie die anderen Schlüsselwörter der Suche im Paragraphen hervorgehoben werden (siehe Kapitel 3.3).

DBpedia Spotlight unterstützt schon mehrere Sprachen bei der Entity Erkennung. Allerdings funktioniert der Entity Service des Lehrstuhls im Moment nur mit englischen Texten.

Aufgerufen wird der Entity Service aus einem AngularJS Service (KeywordService), welcher den erhaltenen Entities noch einen Score zu weißt. Der Score errechnet sich aus der Anzahl der Vorkommen der Entity im Paragraphen und einer Gewichtung der Entity-Typen. Ist der Score größer als ein gegebener Schwellwert, wird die Entity in die Ergebnisliste eingefügt. Der KeywordService, welcher im JarvisBG Modul registriert ist, gibt darauf hin die Liste mit den Ergebnissen zurück.

4.3.3 EuropeanaService

Der EuropeanaService ist für die Kommunikation mit Europeana zuständig. Er erhält die Suchwörter von den Content-Skripten und baut daraus eine Suchanfrage. Die Europeana API akzeptiert boolesche Ausdrücke, also Ausdrücke in denen die Terme mit "AND", "OR" oder "NOT" verknüpft sind. In der beschriebenen Implementierung des EuropeanaServices werden alle Suchwörter mit einem "OR" verknüpft. Zur Steigerung der Ergebnisgüte könnte in Zukunft noch die Möglichkeit geschaffen werden, die Suchanfrage zu verbessern in dem einzelne Suchwörter mit "AND" verbunden werden oder andere mit "NOT" ausgeschlossen werden.

Die Suchterme, die die Content-Skripte an den Service übergeben haben, können auch aus mehreren Wörtern bestehen. Diese sind dann durch Leerzeichen getrennt. Elemente, die mit Leerzeichen separiert sind, verbindet Europeana automatisch mit einem "AND". Um eine eindeutige Syntax zu gewährleisten werden alle Terme in Klammern eingefasst. So kann es nicht zu Fehlinterpretationen kommen, wenn in einer Anfrage "OR"- und "AND"-Terme vorkommen.

Vor dem Absenden werden noch die Einstellungen, die der User über den Optionsdialog definiert hat, geladen und neben der Suchanfrage in die Nachricht an Europeana eingefügt.

4.3.4 MessageService

Es gibt zwei Versionen des MessageServices. Eine ist im Jarvis Modul registriert und eine im JarvisBG Modul. Die Aufgabe des MessageService, welcher Teil von Jarvis ist, ist die Weiterleitung von Nachrichten aus den Content-Skripten an die Background-Skripte. Dazu wird das Message Passing der Chrome API verwendet. Eine Nachricht besteht dabei aus drei Teilen: 1. Die ID der Extension, deren Background-Skripte angesprochen werden sollen, 2. der Name des gewünschten Services und der Methode die in ihm aufgerufen werden soll sowie 3. eine Callback-Methode die vom Empfänger nach der Abarbeitung der Nachricht aufgerufen werden soll.

Der MessageService im JarvisBG Modul hört auf Nachrichten die an die Background-Skripte gesendet werden. Hier werden alle zur Verfügung stehenden Services registriert. Kommt eine Nachricht an, extrahiert der er den Namen des gewünschten Services sowie den Namen der Methode und führt diese aus. Weiterhin stellt er Funktionen zur Kommunikation mit den Content-Skripten in den verschiedenen Browser Tabs bereit.

4.3.5 ParagraphDirective

Die ParagraphDirective umschließt die Paragraphen der Seite. Sie beinhaltet die GUI Elemente zur Manipulation der Schlüsselwörter sowie zur Anzeige der Ergebnisse.

Der Controller der Direktive synchronisiert sich mit dem Chrome Storage um auf Zustandsänderungen der Extension zu reagieren. Deaktiviert der User die Anwendung über das Popup, werden die graphischen Elemente der ParagraphDirective ausgeblendet. Wird sie aktiviert erscheinen sie wieder. Klickt der User auf den Suchbutton, wird eine Anfrage, die den Text des jeweiligen Paragraphen beinhaltet, an JarvisBG gesendet. Dazu wird der schon erwähnte MessageService genutzt. Sobald die Ergebnisse vom KeywordService in der Direktive ankommen, wird eine Suchanfrage mit den gefundenen Schlüsselwörtern über die Background-Skripte an Europeana gesendet. Icons zum Ansehen der Quellen werden bei Ankunft der Ergebnisse neben den gefundenen Schlüsselwörtern angezeigt.

4.3.6 AnchorDirective

Der User soll per Doppelklick Wörter im Text auswählen können und sie so zur Suchanfrage hinzufügen können. Auch das Löschen von Schlagwörtern durch Doppelklick auf hervorgehobene Wörter im Text soll möglich sein. Ein Problem dabei sind Hyperlinks. Bei einem Klick auf einen Anchor Knoten wird sofort der zugewiesene Link aufgerufen. Ein Doppelklick funktioniert also nicht da die neue Seite schon geladen wurde. Um dieses Problem zu umgehen wurde eine neue Direktive entworfen, die das Verhalten von Anchor Tags imitieren soll. Sie übernimmt alle Eigenschaften wie Titel und CSS Klassen vom ursprünglichen Anchor Tag. Die Zieladresse des Hyperlinks wird der neuen AnchorDirektive übergeben und dort gespeichert. Bei einem Klick auf die Direktive wird ein Countdown von 300 ms gestartet. Wird während dieser Zeit kein zweiter Klick registriert lädt die Direktive die Seite des Hyperlinks. Macht der User jedoch einen Doppelklick, verhindert sie, dass sich die Seite ändert.

4.4 Implementierung der GUIs

In diesem Teil wird beschrieben, wie das Konzept des Ramping Interfaces umgesetzt wurde und wie man versucht hat zu Erreichen, dass die zusätzlichen Elemente die Aufmerksamkeit des Users nicht zu sehr belasten. Für die Beschreibung der GUI Elemente wird “Hover” als Beschreibung für den Zustand eines Elements eingeführt, auf welchem sich gerade der Cursor der Maus befindet. Als Farbpalette für die Extension wurde sich für blau und rot entschieden, da sich diese Farben von den meisten Designs gut abheben. So erkennt der Nutzer leicht, welche Elemente zur Extension gehören und welche zur ursprünglichen Webseite.

Abbildung 2: Hervorgehobener Paragraph

Ist die Extension aktiviert und geladen, werden alle gefundenen Paragraphen mit einem Rahmen umschlossen (siehe Abbildung 2). So wird verdeutlicht, was alles zu einem Abschnitt dazugehört. Im oberen linken Eck des Paragraphen befindet sich ein Icon mit dem zuerst die Analyse des Paragraphen-Inhalts gestartet wird und anschließend automatisch eine Anfrage an Europeana gesendet wird. Wurde der Paragraph schon durchsucht. Wird nur noch die Anfrage an Europeana abgeschickt.

Abbildung 3: Hervorgehobener Paragraph im Hover-Zustand

Um den Nutzer nicht unnötig abzulenken, verlieren die Elemente der einzelnen ParagraphDirectives an Deckkraft, wenn sie nicht im Hover Zustand sind, also der Cursor sich nicht über dem Element befindet. Fährt der User mit der Maus über einen Paragraphen, erscheint er wieder in voller Farbe (siehe Abbildungen 2 und 3).

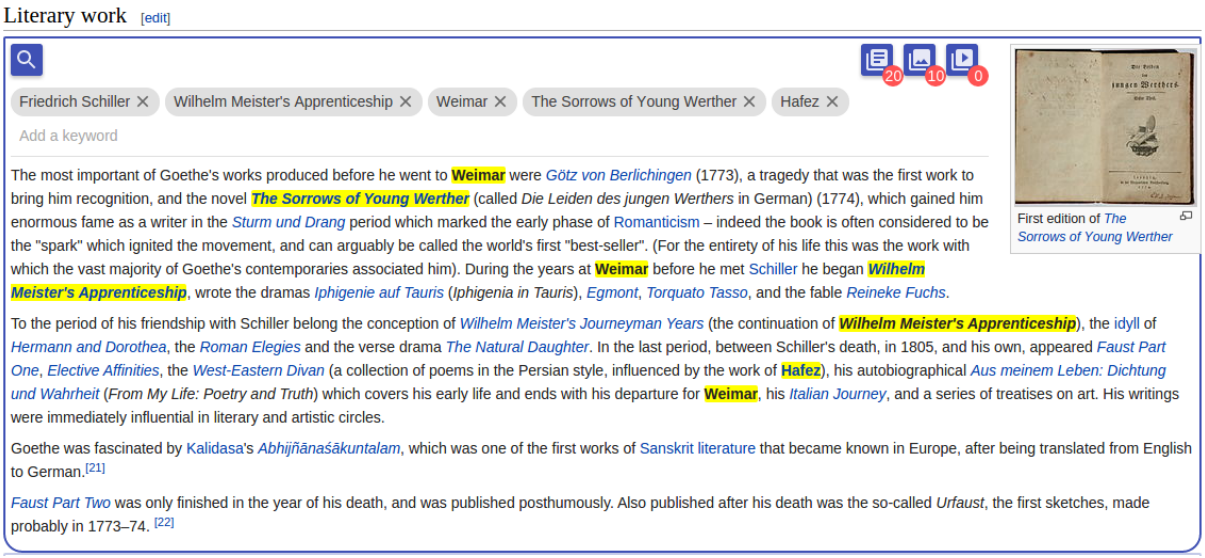

Abbildung 4: Anzeige der gefundenen Keywords sowie Anzahl der Ergebnisse

Sobald Schlüsselwörter zur Verfügung stehen, werden sie im Text hervorgehoben sowie oberhalb des Textes angezeigt (siehe Abbildung 4). Dieser Fall tritt ein, wenn der Nutzer per Doppelklick ein Wort im Text ausgewählt hat oder wenn er durch Betätigen des Knopfes die Analyse des Paragraphen ausgelöst hat. Zur Anzeige der Schlüsselwörter wurden md-chips[18] verwendet. Diese Direktive aus der Angular Material Bibliothek gestattet die einfache Darstellung von Listenelementen und bietet zudem intuitive Steuerungsmöglichkeiten. Durch einen Klick auf das Kreuz neben einem Schlüsselwort kann dieses entfernt werden. Zudem kann mit den Pfeiltasten zwischen den Schlüsselwörtern hin und her gewechselt werden und durch das Betätigen der Entfernen-Taste kann das ausgewählte Element entfernt werden. Die md-chips Direktive bietet zudem eine gute Möglichkeit für die Zukunft, um Relevance Feedback umzusetzen. Darauf wird in Kapitel 6 genauer eingegangen.

Sobald Ergebnisse von Europeana zur Verfügung stehen, erscheinen im oberen rechten Eck des Paragraphen drei Icons, eines für jeden Quellen-Typ. Ein Zähler an jedem Icon signalisiert dem User, wie viele Ergebnisse zum jeweiligen Quellen-Typ gefunden wurden. Nach einem Klick auf eines der Icons öffnet sich ein Dialog mit den Ergebnissen. Das Suchsymbols wird blau um zu verdeutlichen, das die Ergebnisse aktuell sind.

Hat der User die Schlüsselwörter angepasst, wird das Suchsymbol wieder rot und die Icons zur Anzeige der Ergebnisse verschwinden. Durch einen erneuten Klick auf das rote Icon wird eine neue Anfrage losgeschickt ohne den Paragraphen nochmals nach

[18] `https://material.angularjs.org/latest/#/api/material.components.chips/directive/mdChips`

Schlüsselwörtern zu durchsuchen.

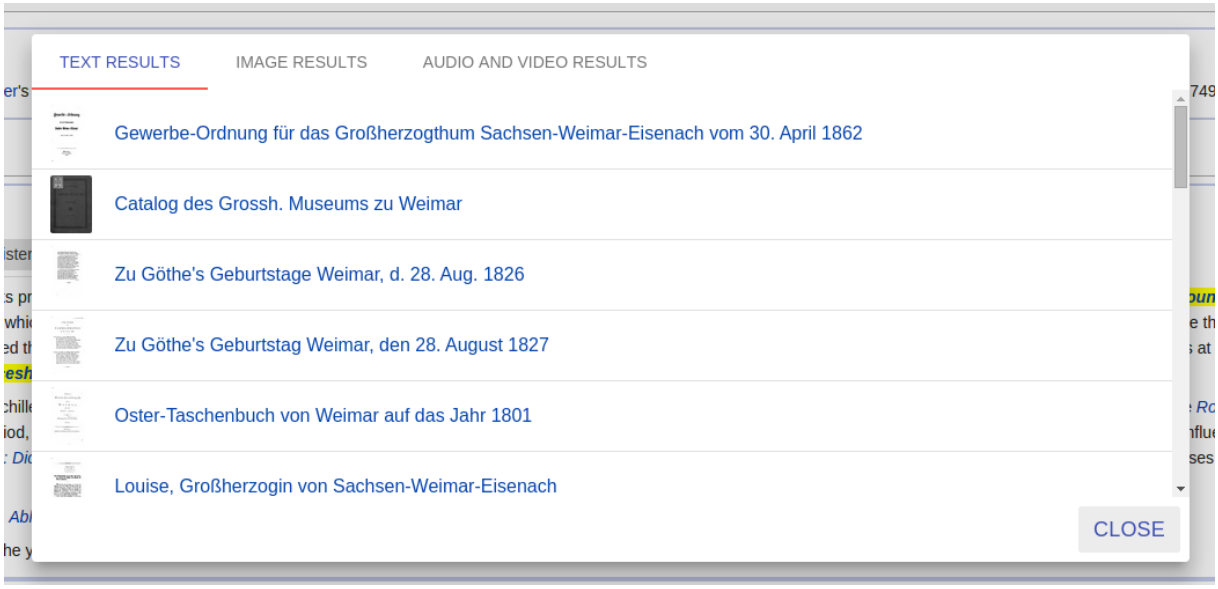

Abbildung 5: Dialog mit gefundenen Textquellen

Abbildung 5 zeigt den Dialog für textuelle Ergebnisse. Mit dem oberen Reiter kann der User zwischen den Quellen-Typen umschalten. Text-Quellen sowie Audio- und Video-Quellen werden in einer Liste dargestellt. Die Listenelemente bestehen aus einem Miniaturbild und dem Titel der Quelle. Durch einen Klick auf das Bild oder den Titel gelangt der der User direkt zu Europeana, wo sich die Ressource befindet. Fährt er mit der Maus über das Miniaturbild wechselt das Element in den Hover-Zustand.

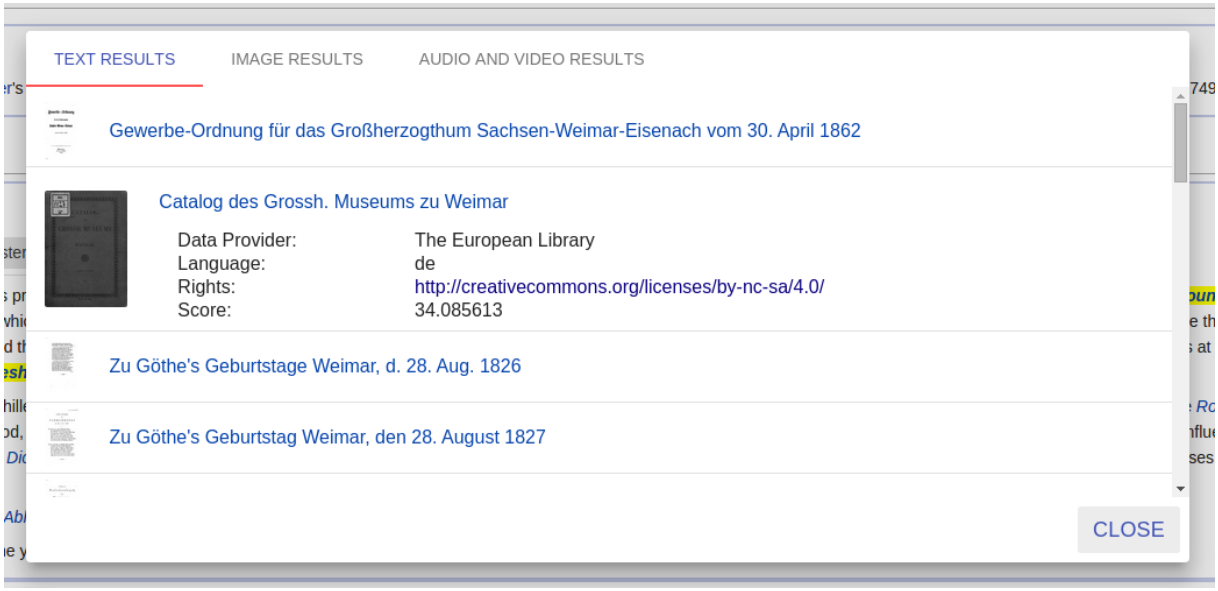

Abbildung 6: Detaillierte Ansicht einer Textquelle

Der Hover-Zustand ist gleichzeitig die letzte Stufe des Ramping Interfaces. Nach der Anzahl der gefundenen Quellen und Miniaturbild und Titel der Quellen werden nun detailliertere Informationen angezeigt. Wie in Abbildung 6 zu sehen ist wird jetzt das Bild der Quelle vergrößert. Außerdem werden unter dem Titel noch der Anbieter der Ressource sowie Sprache und Nutzungsrechte dargestellt. Die Ergebnisse sind nach einem Relevance Score sortiert, den die Europeana API für jede Quellen mitsendet. Je höher der Score umso höher die Wahrscheinlichkeit, dass die Ressource für den User interessant ist. Dieser Wert wird ebenfalls angezeigt.

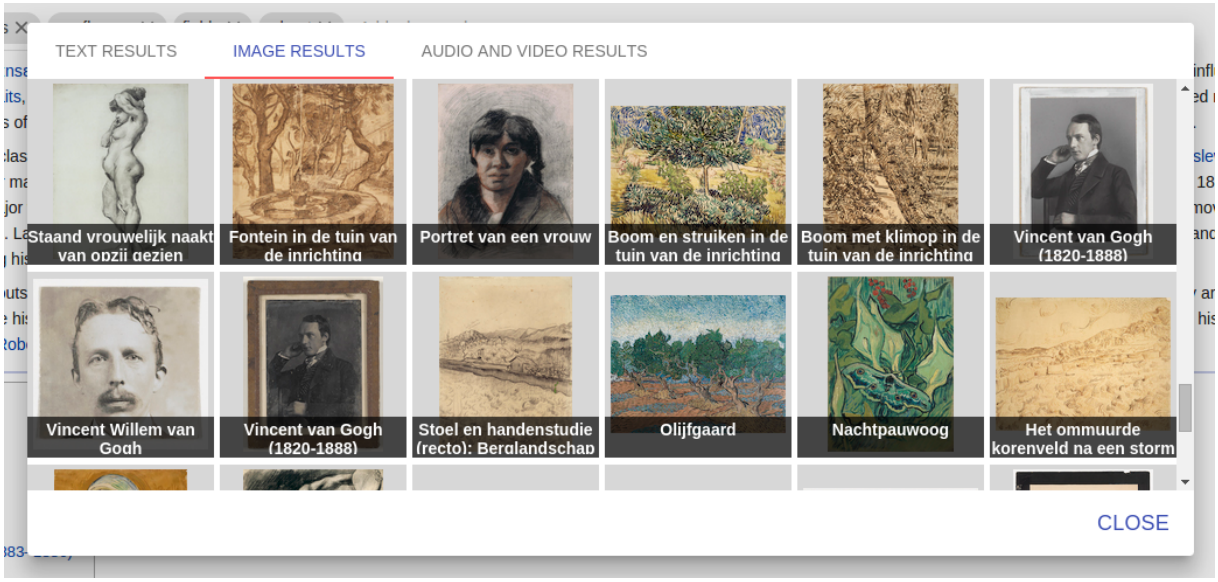

Abbildung 7: Dialog mit gefundenen Bildquellen

Bei Bildquellen sind Metadaten wie die Sprache der Ressource nicht so Relevant wie das eigentliche Bild. Aus diesem Grund wurde für die Anzeige der Bildquellen eine andere Darstellungsweise gewählt. Die Ergebnisse werden in einer md-grid-list[19] angezeigt. Die Elemente der Liste bestehen aus dem Bild sowie dem Titel. Auch hier gelangt der User durch einen Klick direkt zur Ressource, jedoch existiert kein separater Hover-Zustand.

Ist ein Vorschaubild nicht verfügbar oder wurden keine Quellen des ausgewählten Typs gefunden, wird stattdessen ein Platzhalter an der entsprechenden Stelle angezeigt.

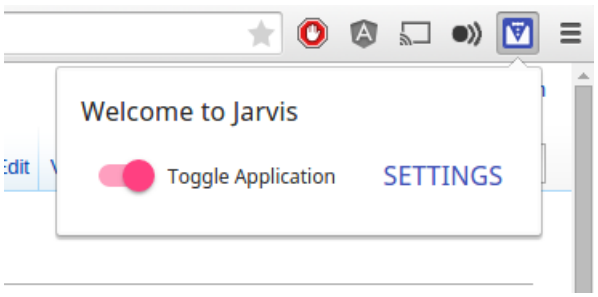

Abbildung 8: Popup zum aktivieren der Extension

Um die Extension zu aktivieren oder deaktivieren muss der User auf das Jarvis-Icon neben der Adressleiste des Browsers klicken. Dadurch öffnet sich ein Popup (siehe Abbildung 8). Durch einen Klick auf den Umschalter wechselt die Anwendung ihren Zustand. Allerdings

[19] `https://material.angularjs.org/latest/#/api/material.components.gridList/directive/mdGridList`

gilt dies nur für den aktuellen Tab. Die Jarvis Module in anderen Tabs sind dadurch nicht betroffen. Weiterhin bietet das Popup dem User einen Knopf an, mit dem er zum Einstellungsdialog gelangt.

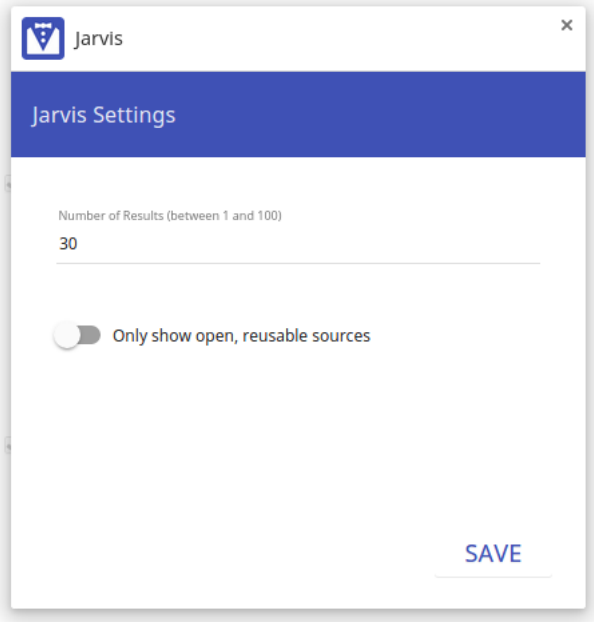

Abbildung 9: Dialog zum anpassen der Einstellungen

Die Einstellungen der Extension können über ein separates Fenster angepasst werden. Der in Abbildung 9 dargestellte Dialog ist über den Extension Manager[20], sowie über den erwähnten Knopf im Popup, erreichbar. Er bietet momentan die Möglichkeit die Anzahl der Ergebnisse fest zu legen, die von Europeana geliefert werden sollen. Auch kann der User entscheiden ob er alle Quellen sehen möchte, oder nur solche, die er weiterverwenden und bearbeiten darf.

4.5 1. Erweiterung des Prototyps

Nach der ersten evaluation design und verhalten angepasst. Optionen hinzugefügt. Button entfernt

4.6 2. Erweiterung des Prototyps (optional)

Nach der zweiten evaluation neues recommender system angebunden

[20]Innerhalb von Chrome zu finden unter: `chrome://extensions/`

5 Evaluation

5.1 1. Evaluation: Interne Evaluation

5.2 2. Evaluation: Thinking Aloud Tests

Um die Benutzbarkeit der Anwendung zu testen wurden Thinking Aloud Tests durchgeführt. Bei dieser Art der Evaluation werden den Testern Aufgaben gestellt, die sie mit Hilfe der Anwendung lösen müssen. Dabei sollen sie alles was sie denken und alle Entscheidungen die sie treffen laut aussprechen. Sie sollen sagen warum sie auf genau diesen Knopf drücken oder welche Fragen sich ihnen bei der Nutzung stellen. So können Probleme in der Benutzbarkeit aufgedeckt werden.

Vorteile dieser Methode sind, dass nur eine geringe Anzahl von Testern (in diesem Fall 4) gebraucht werden. Sie kann schon in einem frühen Stadium der Entwicklung mit einem Prototypen durchgeführt werden[21] und hilft dabei, Designfehler früh zu erkennen und zu beheben. Jedoch ist das Feedback der Testuser sehr subjektiv und das Sprechen während des Testens verlangsamt den Benutzer. Da hier die Benutzbarkeit des Systems getestet werden soll - und nicht etwa die Effizienz - sind Thinking Aloud Tests gut geeignet.

5.2.1 Testaufbau

- einführung - jeder tester einzeln - verschiedene Wissenslevel - filmen - Abschließender Fragebogen - aufbau (hardware, display, maus, tastatur, display) - dataagreement - personen beschreiben - personen einzeln auswerten (anhang, informell, anonym). hier nur aggregation - Fragebogen und Zettel in den Anhang.

5.2.2 Auswertung

5.3 3. Evaluation: Expertentest (optional)

Nach Einbindung des neuen Recommenders nochmal durch Experten getestet ob es sich verbessert hat.

[21] Bei der Evaluation von Jarvis war die Implementierung schon abgeschlossen

6 Future Work

6.1 Verbesserung der Ergebnis-Güte

6.1.1 Speicherung von guten Suchanfragen für jeden Paragraphen

6.1.2 Bewertung von Quellen

6.1.3 Automatic Query Extension

Re-examining the Potential Effectivness of Interactive Query Expansion Query expansion techniques, e.g. [1,5] aim to improve a user's search by adding new query terms to an existing query. A standart method of performing query expansion is to use relevance information from the user All or some of these expansion terms can be added to the query either by the user - interactive query expansion - or by the retrieval system - automatic query expansion. argument for IQE is that interactive query expansion gives more control to the user We compare the effect of query expansion against no query expansion. how good are different approaches to query expansion? All AQE strategies were more likely on average to improve a query than harm it. All techniques improved at least 50% of the queries where query expansion could make a difference to retrieval effectivness. query dependet strategy not only improves the highest percentage of queries, is most stable, but also gives the highest average precision over the queries, is most effective users, especially web searchers, often use very short queries [9]. presenting a list of possible expansion terms is one way to get users to give more information, in the form of query words, to the system. people can recognize expansion terms that are semantically related to the informatin for which they are seeking and expand the query using these terms

6.1.4 Interactive Query Extension - relevance feedback

Towards Interactive Query Expansion: in an era of online retrieval, it is appropriate to offer guidance to users wishing to improve their initial queries. one form of such guidance could be short lists of suggested terms gathered from feedback, nearest neigbhors, and term variants of original query terms one method for retrieving more relevant documents is to expand the query terms by using relevance feedback, conflating word stems, and/or adding synonyms form thesaurus. These additional terms allow the query to match documents that contain words which are related to the query, but not actually expressed in it. system should be able to help the user modify the query in order to retrieve more relevant documents The size of windows and the patience of the user require that only a reasonable number of terms be displayed relevance feedback is an interactive retrieval tool with the user selecting terms form a small subset of the terms derived from relevance feedback In an interactive situation, the user will select only those terms deemed useful it has further been shown that user selection from term vaiants and nearest neighbors of query terms

can provide terms for query expansion that improve performance to that comparable with feedback.

6.1.5 word sense disambigation (mal schauen)

[BH00]

6.1.6 Filtern der Ergebnisse, clustering

(mehr Präzision da Ausbeute bei JITIR nicht so relevant) Clustering[BH00]

6.2 Anbindung weiterer Quellen (nur wenn nicht Teil der Implementierung)

Automatische Auswahl der richtigen QUelle laut selecting task relevant sources

6.3 Portierung der Anwendung auf andere Plattformen

6.3.1 Anpassung der Anwendung für mobile Nutzung

6.4 Beseitigung der Einschränkungen

Overlay bauen, whitlistings implementieren

7 Conclusion

- Steigerung der Effektivität und Produktivität von wissenschaftlichem Arbeiten
- Starke Effektivitätssteigerung durch Punkte aus Future Work möglich?
- information management assistants embody a vision of a future in which users hardly ever form a query to request information. when an information need arises, a system like watson has already anticipated it and provided relevant information to the user before she is even able to ask for it (budzik, watson)
- survey suggested that users are relatively dissatisfied with the results of their searching experience. THis makes concrete the claim that systems designed to help useres in their information seeking tasks are needed in the world (budzik, watson)

  JITIR's can be thought of as automatic "query free" search engines (rhodes, using physical context)

Limitations: zB manche seiten werden nicht schön angezeigt (auch beim deaktivieren nicht) nicht alle keywords werden angezeigt Durchschinittle länge der paragraphen sehr unterschiedlich - ¿service der damit klar kommt

8 Quellenverzeichnis

[AI] Alexa Internet, Inc.: *Alexa*. http://www.alexa.com/topsites. – Zugriff: 13.05.2015, Archiviert mit WebCite®: http://www.webcitation.org/5hgzUZacN

[And97] Anderson, Kenneth M.: Integrating open hypermedia systems with the World Wide Web. In: *Proceedings of the eighth ACM conference on Hypertext* ACM, 1997, S. 157–166

[BH99] Budzik, Jay ; Hammond, Kristian: Watson: Anticipating and contextualizing information needs. In: *Proceedings of the Annual Meeting-American Society for Information Science* Bd. 36 Citeseer, 1999, S. 727–740

[BH00] Budzik, Jay ; Hammond, Kristian J.: User interactions with everyday applications as context for just-in-time information access. In: *Proceedings of the 5th international conference on intelligent user interfaces* ACM, 2000, S. 44–51

[Bou99] Bouvin, Niels O.: Unifying strategies for Web augmentation. In: *Proceedings of the tenth ACM Conference on Hypertext and hypermedia: returning to our diverse roots: returning to our diverse roots* ACM, 1999, S. 91–100

[Día12] Díaz, Oscar: Understanding web augmentation. In: *Current Trends in Web Engineering*. Springer, 2012, S. 79–80

[DJHM13] Daiber, Joachim ; Jakob, Max ; Hokamp, Chris ; Mendes, Pablo N.: Improving efficiency and accuracy in multilingual entity extraction. In: *Proceedings of the 9th International Conference on Semantic Systems* ACM, 2013, S. 121–124

[GW99] Goldman, Roy ; Widom, Jennifer: Interactive query and search in semistructured databases. In: *The World Wide Web and Databases*. Springer, 1999, S. 52–62

[Har88] Harman, Donna: Towards interactive query expansion. In: *Proceedings of the 11th annual international ACM SIGIR conference on Research and development in information retrieval* ACM, 1988, S. 321–331

[JMM15] Jain, Nilesh ; Mangal, Priyanka ; Mehta, Deepak: AngularJS: A Modern MVC Framework in JavaScript. In: *Journal of Global Research in Computer Science* 5 (2015), Nr. 12, S. 17–23

[JSS00] Jansen, Bernard J. ; Spink, Amanda ; Saracevic, Tefko: Real life, real users, and real needs: a study and analysis of user queries on the web. In: *Information processing & management* 36 (2000), Nr. 2, S. 207–227

[Kah] Kahle, Christian: *Suchmaschinen: Bing arbeitet effizienter als Google.* `http://winfuture.de/news,64930.html`. – Zugriff: 22.05.2015, Archiviert mit WebCite®: `http://www.webcitation.org/6YiTPLrEh`

[Law00] Lawrence, Steve: Context in web search. In: *IEEE Data Eng. Bull.* 23 (2000), Nr. 3, S. 25–32

[Lev] Levy, Steven: *The never ending search.* `https://medium.com/backchannel/how-google-search-dealt-with-mobile-33bc09852dc9`. – Zugriff: 22.05.2015

[Mil68] Miller, Robert B.: Response time in man-computer conversational transactions. In: *Proceedings of the December 9-11, 1968, fall joint computer conference, part I* ACM, 1968, S. 267–277

[MJGSB11] Mendes, Pablo N. ; Jakob, Max ; García-Silva, Andrés ; Bizer, Christian: DBpedia spotlight: shedding light on the web of documents. In: *Proceedings of the 7th International Conference on Semantic Systems* ACM, 2011, S. 1–8

[PS] Prud'hommeaux, Eric ; Seaborne, Andy: *SPARQL Query Language for RDF.* `http://www.w3.org/2001/sw/DataAccess/rq23/`. – Zugriff: 19.05.2015

[Rho00a] Rhodes, Bradley J.: Margin notes: Building a contextually aware associative memory. In: *Proceedings of the 5th international conference on Intelligent user interfaces* ACM, 2000, S. 219–224

[Rho00b] Rhodes, Bradley J.: *Just-in-time information retrieval*, Massachusetts Institute of Technology, Diss., 2000

[RM00] Rhodes, Bradley J. ; Maes, Pattie: Just-in-time information retrieval agents. In: *IBM Systems journal* 39 (2000), Nr. 3.4, S. 685–704

[RS96] Rhodes, Bradley ; Starner, Thad: Remembrance Agent: A continuously running automated information retrieval system. In: *The Proceedings of The First International Conference on The Practical Application Of Intelligent Agents and Multi Agent Technology*, 1996, S. 487–495

[RSBS08] Russell, Alistair ; Smart, Paul R. ; Braines, Dave ; Shadbolt, Nigel R.: Nitelight: A graphical tool for semantic query construction. (2008)

[Rut03] Ruthven, Ian: Re-examining the potential effectiveness of interactive query expansion. In: *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval* ACM, 2003, S. 213–220

[SSG14] Schlötterer, Jörg ; Seifert, Christin ; Granitzer, Michael: Web-based Just-In-Time Retrieval for Cultural Content. In: *Patch '14: proceedings of the 7th international ACM workshop on personalized access to cultural heritage*, 2014

[SSL+15] Seifert, Christin ; Schlötterer, Jörg ; Loehden, Aenne ; Sciascio, Cecilia di ; Tschinkel, Gerwald ; Sabol, Vedran: EEXCESS Deliverable D2.3. – First Usability Evaluation Report / Universität Passau. 2015 (Deliverable D2.3.). – Forschungsbericht

[VK] Valentine, Charles ; Knezevic, Milica: *Hunting for First World War data - CENDARI and the Europeana API.* `http://labs.europeana.eu/blog/CENDARI-and-the-Europeana-API/`. – Zugriff: 22.05.2015, Archiviert mit WebCite®: `http://www.webcitation.org/6YiBa1IBQ`

Anhang

Thinking Aloud Test Anweisungen

Jarvis: Thinking Aloud Evaluation

Was ist Jarvis:

Jarvis ist eine Chrome Browser Extension die eine Webseite in Paragraphen aufteilt und auf Benutzeranfrage weiterführende Quellen mit kulturellen Inhalten anzeigt.

Was sollst du machen:

- Löse die gestellten Aufgaben mit Hilfe von Jarvis
- Rede während dessen

Worüber sollst du reden:

- Was versuchst du zu erreichen?
- Was liest du gerade?
- Was für Fragen stellen sich dir bei der Nutzung?
- Was findest du verwirrend oder störend?
- Welche Entscheidungen triffst du?

Aufgaben:

- Gehe auf en.wikipedia.org
- Aktiviere Jarvis für diesen Tab
- Versuche Quellen zu Schillers Jugend zu finden
- Finde zwei Wege neue Suchwörter hinzuzufügen
- Finde zwei Wege Suchwörter zu entfernen
- Versuche die Sprache einer Quelle heraus zu finden, ohne sie zu öffnen
- Suche im zweiten Absatz von “Early life and career” nach Quellen zu seiner Flucht nach Weimar 1782. Passe dazu ggf. die Suchanfrage so an, dass das gewollte Ergebnis angezeigt wird
- Setze die Anzahl der zurück kommenden Ergebnisse auf 100
- Setze die Sprache der zurück kommenden Ergebnisse auf Englisch

Thinking Aloud Fragebogen

Jarvis: Thinking Aloud Evaluation Fragebogen

1. Jarvis lässt sich intuitiv bedienen
(1: Trifft gar nicht zu - 5: Trifft voll zu) (eine (1) Option auswählen)

○ 1 ○ 2 ○ 3 ○ 4 ○ 5

2. Ich bräuchte eine Anleitung um die Extension wirklich zu verstehen (1: Ohne Anleitung finde ich mich gar nicht zurecht - 5: Alle Funktionalitäten waren intuitiv nutzbar) (eine (1) Option auswählen)

○ 1 ○ 2 ○ 3 ○ 4 ○ 5

3. Wenn du eine der letzten Fragen mit 1 oder 2 beantwortet hast, schreibe bitte kurz warum

_____
_____
_____
_____
_____

4. Die Extension hat für die einzelnen Paragraphen die wichtigen Schlüsselwörter gefunden (1: Trifft gar nicht zu - 5: Trifft voll zu) (eine (1) Option auswählen)

○ 1 ○ 2 ○ 3 ○ 4 ○ 5

5. Die gefundenen Quellen waren relevant (1: Trifft gar nicht zu - 5: Trifft voll zu) (eine (1) Option auswählen)

○ 1 ○ 2 ○ 3 ○ 4 ○ 5

6. Das Design der Anwendung ist ansprechend
(1: Trifft gar nicht zu - 5: Trifft voll zu) (eine (1) Option auswählen)

○ 1 ○ 2 ○ 3 ○ 4 ○ 5

7. Das Design der Anwendung lenkt mich nicht von meiner eigentlichen Arbeit ab (1. Trifft gar nicht zu - 5: Trifft voll zu) (eine (1) Option auswählen)

○ 1 ○ 2 ○ 3 ○ 4 ○ 5

8. Ich würde die Anwendung weiterhin nutzen
(1: Trifft gar nicht zu - 5: Trifft voll zu) (eine (1) Option auswählen)

○ 1 ○ 2 ○ 3 ○ 4 ○ 5

9. Hast du Anmerkungen oder Verbesserungsvorschläge?

_____
_____
_____
_____
_____

Erklärung

Hiermit versichere ich, dass ich meine Abschlussarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Datum: .....

(Unterschrift)